CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1 – Sep. 2019

Sub:	Programming Using C# and .Net							Sub Code:	17MCA 52
Date:	21/09/2019	Duration:	90 min's	Max Marks:	50	Sem	5th	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE CO	OBE RBT
1	Explain the Benefits and architecture of .NET framework.	[10]	CO1	L1

The latest version of .NET framework provides:

- In developing portable, scalable, and robust applications.
- Developed applications can be executed in a distributed environment.

1.4 Components of .NET Framework 4.0:

The .NET Framework provides all the necessary components to develop and run an application. The components of .NET Framework 4.0 architecture are as follows:

- Common Language Runtime (CLR)
- Common Type System (CTS)
- Metadata and Assemblies
- .NET Framework class library
- Windows Forms
- ASP.NET and ASP.NET AJAX
- ADO.NET
- Windows Workflow Foundation
- Windows Presentation Foundation
- Windows Communication Foundation
- Windows CardSpace
- LINQ

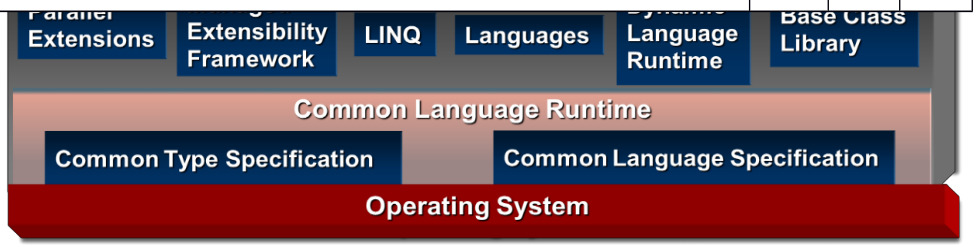
Let's now discuss about each of them in detail.

1.5 CLR[Common Language Runtime]:

"CLR *is an* ***Execution Engine for .NET Framework applications*****".**

CLR is a heart of the.NET Framework. It provides a run-time environment to run the code and various services to develop the application easily.

The services provided by CLR are –



- Memory Management
- Exception Handling
- Debugging
- Security
- Thread execution
- Code execution
- Language Integration
- Code safety
- Verification
- Compilation

The following figure shows the **process** of compilation and execution of the code by the JIT Compiler:

i. After verifying, a **JIT** [*Just-In-Time*] compiler extracts the metadata from the file to translate that verified IL code into ***CPU-specific code*** or ***native code***. These type of IL Code is called as **managed code**.

ii. The source code which is directly compiles to the machine code and runs on the machine where it has been compiled such a code called as **unmanaged code**. It does not have any services of CLR.

iii. Automatic garbage collection, exception handling, and memory management are also the responsibility of the CLR.

Managed Code: Managed code is the code that is executed directly by the CLR. The application that are created using managed code automatically have CLR services, such as type checking, security, and automatic garbage collection.

The process of executing a piece of managed code is as follows:

- Selecting a language compiler
- Compiling the code to IL[This intermediate language is called managed code]
- Compiling IL to native code Executing the code

Unmanaged Code: Unmanaged Code directly compiles to the machine code and runs on the machine where it has been compiled. It does not have services, such as security or memory management, which are provided by the runtime. If your code is not security-prone, it can be directly interpreted by any user, which can prove harmful.

Automatic Memory Management: CLR calls various predefined functions of .NET framework to allocate and de-allocate memory of .NET objects. So that, developers need not to write code to explicitly allocate and de-allocate memory.

1.6 CTS [Common Type Specifications]:

The CTS defines the rules for declaring, using, and managing types at runtime. It is an integral part of the runtime for supporting cross-language communication.

The common type system performs the following functions:

- Enables cross-language integration, type safety, and high-performance code execution.

	🗣 Provides an object-oriented model for implementation of many programming languages. 🗣 Defines rules that every language must follow which runs under .NET framework like C#, VB.NET, F# etc. can interact with each other. The CTS can be classified into two data types, are i. Value Types ii. Reference Type			
2	What is an Assembly? Describe the information stored in assembly manifest by differentiating and Multiple assemblies. **Assemblies can stored in two types:** **Static assemblies:** Static assemblies include interfaces, classes and resources. These assemblies are stored in PE (Portable executable) files on a disk. **Dynamic assemblies:** Dynamic assemblies run directly from the memory without being saved to disk before execution. However, after execution you can save the dynamic assemblies on the disk. **Global Assembly Cache:** The Global Assembly Cache (GAC) is ***a folder in Windows directory*** to store the .NET assemblies that are specifically designated to be shared by all applications executed on a system. ➢ The assemblies must be sharable by registering them in the GAC, only when needed; otherwise, they must be kept private. ➢ Each assembly is accessed globally without any conflict by identifying its name, version, architecture, culture and public key. You can deploy an assembly in GAC by using any one of the following: 🗣 An installer that is designed to work with the GAC 🗣 The GAC tool known as **Gacutil.exe** 🗣 The Windows Explorer to drag assemblies into the cache. **Strong Name Assembly:** A Strong Name contains the assembly's identity, that is, the information about the assembly's name, version number, architecture, culture and public key. 🗣 Using Microsoft Visual Studio .NET and other tools, you can provide a strong name to an assembly. 🗣 By providing strong names to the assembly, you can ensure that assembly is globally unique.	[10]	CO1	L2

	Private and Shared Assembly: A single application uses an assembly, then it is called as **a private assembly.** Example: If you have created a DLL assembly containing information about your business logic, then the DLL can be used by your client application only. Therefore, to run the application, the DLL must be included in the same folder in which the client application has been installed. This makes the assembly private to your application. Assemblies that are placed in the Global Assembly cache so that they can be used by multiple applications, then it is called as a **shared assembly**. Example: Suppose the DLL needs to be reused in different applications. In this scenario, instead of downloading a copy of the DLL to each and every client application, the DLL can be placed in the global assembly cache by using the **Gacutil.exe** tool, from where the application can be accessed by any client application. **Side-by-Side Execution Assembly:** The process of executing **multiple versions** of an *application* or an *assembly* is known as **side-by-side execution**. Support for side-by-side storage and execution of different versions of the same assembly is an integral part of creating a strong name for an assembly. - Strong naming of .NET assembly is used to provide unique assembly identity by using the sn.exe command utility. - The strong-named assembly's version number is a part of its identity, the runtime can store multiple versions of the same assembly in the GAC. - Load these assemblies at runtime.			
3	With a neat diagram explain the workflow of .NET execution engine. The following figure summarizes the workflow between a .NET source code, a .NET compiler, and the .NET execution engine:	[10]	CO1	L2

	The .NET execution process completes as given below: iii. When you compile source code by selecting **.NET aware** compilers such as Visual Basic, C#, Visual C++, J#, or any of the third party compilers, such as COBOL, Perl or Eiffel. iv. The .Net aware compiler converts source code in to binaries that are called as **assemblies.** The assembly can be either ***.dll** or ***.exe** depending on the entry point defined in the application. v. Assembly contains IL code, Metadata and Manifest data. **NOTE: IL**(Intermediate Language) code is also known as **MSIL**(Microsoft IL) / **CIL**(Common IL) has a **machine-readable instruction sets**. vi. Then loaded IL code must be converted to Platform-specific code by a **Just-in-Time(JIT) compiler** at runtime. vii. **Base class Library (mscorlib.dll):** This library encapsulates various primitives such as file IO, Data Access, Threading, XML/SOAP etc. When building .NET binaries you always make use of this particular assembly.			
4	Bring out the difference between value types and reference types and also write a program for boxing and unboxing.	[10]	CO1	L2

Vale Types	Reference Types
Allocated on stack	Allocated on heap
variable contains the data itself	variable contains the address of memory location where data is actually stored
When we copy a value type variable to another one, the actual data is copied and each variable can be independently manipulated.	When copying a reference type variable to another variable, only the memory address is copied. Both variables will still point to the same memory location, which means, if we change one variable, the value will be changed for the other variable too.
Integer, float, boolean, double etc are value types.	string and object are reference types.
Derived from System.ValueType	Derived from System.Object

5 Write a c# program to explain accessor and mutator used in Encapsulation. [10] CO2 L3

Rather than defining the data in the form of *public*, we can declare those fields as ***private*** so that we achieved encapsulation. The Private data are manipulated using ***accessor*** **(get: store the data to private members)** and ***mutator*** **(set: to interact with the variable)** methods.

Syntax:

```
set { <accessor-body> }
get { <accessor-body> }
```

Note: Keep in mind about property

- Although you don't have to have both a getter and a setter, you must have one of the two.
- A property defined with both a *getter* and a *setter* is called a **read-write** property.
- A property defined with only a *getter* is called a **read-only property**.
- A property defined with only a *setter* is called a **write-only property**.

The Private data are manipulated indirectly by **two** ways.

i. Traditional accessor and mutator methods. **ii. Named property**

i. The **first method** is, if we want the outside world to interact with private *usn* data field, **tradition**

	dictates defining an *accessor* (get method) and *mutator* (set method).
	Example 2.1: In this application, we have defined two methods `set()` and `get()`. The `set()` method mutator, set the value of `usn` variable. The `get()` method accessor, displays the value of the `usn` variable on the command prompt.
	`using System;`
	`namespace Chapter4_Examples { class Student {`
	`string name, branch,usn;`
	`public void setusn(string sid){ usn = sid;`
	`}`
	`public string getusn(){ return usn;`
	`}`
	`}`
	`class GetSetDemo { static void Main(){`
	`Student st1 = new Student();`
	`st1.setusn("1RX12MCA01"); Console.WriteLine("USN: "`
	`+st1.getusn()); Console.ReadKey();`
	`}`
	`}`
	`}`			

6	How do you prevent inheritance using sealed classes? Explain with an Example.
	The next pillar of OOP, Inheritance provides you to reuse existing code and fast implementation time. The relationship between two or more classes is termed as ***Inheritance***.
	In essence, inheritance allows to extend the behavior of a base (or parent/super) class by enabling a subclass to inherit core functionality (also called a derived class/child class). All public or protected variables and methods in the base class can be called in the ***derived classes.***
	Inheritance comes in two ways:
	- Classical inheritance ("is-a" relationship)
	- Containment/delegation model ("has-a" relationship).
	`using System;`
	`namespace Chapter4_Examples{ class Animal`
	`{`
	`public Animal(){ Console.WriteLine("Base class constructor");`
	`}`
	`public void Greet(){ Console.WriteLine("Hello,I am kind of Animal");`
	`}`
		[10]	CO2	L3

```
   }
   class Dog : Animal{ public Dog()
    {
      Console.WriteLine("Derived class constructor");
    }
}
 class isademo{ static void Main(){
     Dog d = new Dog(); d.Greet(); Console.ReadKey();
  }
 }
 }
```

Sealed Class:-

Sealed classes are classes that cannot be inherited. You can use the ***"sealed"*** keyword to define a class as a sealed class.

Syntax:

```
sealed class <Class_name>{
    // data mebers and member functions
}
```

```
sealed class MyClass{
      . . . // data mebers public void GetDetail(){
            //Code
      }
      public void ShowDetail(){
            //Code
      }
}
 Class MainClass{
      //Instantiation of myclass class
      //Method calling
}
```

7	Explain partial Classes and Partial Methods with the help of a program.	[10]	CO2	L3
8	What is Polymorphism? Write a c# program to explain Method Overloading. The final pillar of OOP is **polymorphism**. Polymorphism means ***"one name many forms"***. Polymorphism defined as ***"one object behaving as multiple forms and one function behaves in different forms"***. In other words, ***"Many forms of a single object is called Polymorphism"***. **Advantages:**	[10]	CO2	L3

- Allows you to invoke methods of a derived class through base class reference during runtime
- Provides different implementations of methods in a class that are

called through the same name There are **two types** of polymorphism, which

are as follows:

i. Static polymorphism/Compile Time polymorphism/Early Binding/Overloading
ii. Dynamic polymorphism/Run-time polymorphism/Late Binding/Overriding

i. Method overloading:

In method overloading, can be define many methods with the same name but different signatures. A method signature is the combination of the method's name along with the number, type, and order of the parameters.

Example 3.1: In this application, the `Area()` method of Shape class is overloaded for calculating the area of `square, rectangle, circle` and `triangle` shapes. In the `Main()` method, the `Area()` method is called multiple times by passing different arguments

```
using
System;
namespace
Class_Dem
os{
    class Shape{
        public void Area(int side){
          Console.WriteLine("The area of Square is: " +
          side * side);
        }
        public void Area(int length, int
          breadth){ Console.WriteLine("The
          area of Rectangle is: " + length
          * breadth);
}

        public void Area(double radius)
            { Console.WriteLine("The area of
            Circle is: "
                                        + 3.14 * radius *
                                        radius);
        }
        public void Area(double base1,
```

```
            double height)
            { Console.WriteLine("The area of
            Squate is: "
                                                + (base1 *
                                                height)/2);
        }
    }
    class
        MOverloa
        d{ stati
        c void
        Main(){
  Shape shape = new Shape();
   shape.Area(15);
  shape.Area(10, 20);
  shape.Area(10.5);
  shape.Area(15.5, 20.4);
  Console.Read();
        }
    }
}
```

9	What is Operator Overloading? Write a c# program to explain Operator Overloading. **i. Operator Overloading:** The mechanism of assigning a special meaning to an operator according to user defined data type, such as classes or structs, known as ***operator overloading***. The below table shows the list of operators and overloading status.	[10]	CO2	L3

```
using
System;
namespace
Class_Demos
{
    clas
        s

        u
        n
        a
        r
        y
        o
        p
```

```
        r
        {

        i
        n
        t

        n
        1
        ,

        n
        2
        ;
        public unaryopr() { }
        public unaryopr(int
            a, int b){ n1 =
            a;
            n2 = b;
        }
        public void showData(){
           Console.WriteLine("The numbers are: " +n1+ "
           and " +n2);
        }
        public static unaryopr operator -
            (unaryopr opr){ unaryopr obj =
            new unaryopr();
            obj.n1
            =
            -opr.n
            1;
            obj.n2
            =
            -opr.n
            2;
            return
            obj;
        }
    }
    class
        OpOverlo
        ad{ stat
        ic void
        Main(){
```

```
            unaryopr opr1 = new unaryopr(20,30);
            Console.WriteLine("Before Operator
            Overloading"); opr1.showData();
            unaryopr opr2 = new unaryopr();
            opr2 = -opr1;   //invoke operator
            overloading method
            Console.WriteLine("After Operator
            Overloading"); opr2.showData();
            Console.Read();
        }
    }
}
```

10 Write a C# program to Demonstrate Use of Static class and Static Method. [10] CO2 L3

Static class:

When a **class** has been defined as ***static***, no need to create an object. A static class must contain only static members, except for constants (if this is not the case, you receive compiler errors).

Main benefit: We do not need to make any instance of this class; all members can be accessible with its own name.

```
 using
System;
namespace
Examples {
 static class StClass
 {
 static int record = 0;
 static void printrecord(){
 Console.WriteLine("No of stud record: {0}", record);
 }
 static void Main()
{ record = 2;
printrecord();
Console.ReadLine();
 }
 }
 }
```